Follow-up Assignment:

Encoding and Persistent Storage of Graphical Data in Java

Part 1

November 18, 2025

1. Preparing a class for encoding coordinates and color. Create the class `PositionAndColor.java`, and then:

   - declare a public static method named `encode`, which takes three arguments: two integers `x` and `y` representing coordinates, and an object of type `Color` representing the color, and returns an `int` value serving as an encoded data structure.
     - Inside the method, create a local variable `result` of type `int` and assign to it the value computed from the parameter `x`, retaining only the lowest 12 bits of this coordinate using the bit mask `0xFFF`, so that other bits do not affect the result.
     - Extend the contents of the variable `result` with the coordinate `y` by taking the lowest 12 bits of this value using the mask `0xFFF`, then shifting them left by 12 bits and combining them with the current result using the bitwise OR operation, so that `x` and `y` occupy disjoint parts of the integer.
     - Add color information to the encoded value by calling the helper function `colorToByte` with the argument `color`, masking the obtained byte with `0xFF`, then shifting this result left by 24 bits and combining it with the current value of `result` using the bitwise OR operation, so that the byte representing the color is placed in the highest byte of the integer.
     - Return the value of the variable `result` from the defined method as the final encoded result containing in a single integer the information about the coordinates `x`, `y` and the color.
   - Declare a public static method named `decode`, which takes a single parameter of type `int` named `value`, representing an encoded value containing coordinates and color information, and returns an array of type `int`.
     - Extract the coordinate `x` by computing from the parameter `value` the lower 12 bits using the mask `0xFFF` and assign the result to a local variable `x` of type `int`, so as to obtain the original horizontal coordinate.
     - Extract the coordinate `y` by shifting the value `value` right by 12 bits, then applying the mask `0xFFF` to the obtained result, and assign this value to a local variable `y` of type `int`, to recover the vertical coordinate stored in the middle bits.

  - Extract the color information by shifting the value `value` right by 24 bits, then applying the mask `0xFF`, and store the result in a local variable `colorByte` of type `int`, so as to obtain the color byte located in the highest byte of the integer.
  - Return from the method a new integer array containing three elements in the fixed order: `x`, `y`, and `colorByte`, so that the caller may read both the coordinates and the color information.

- Declare a public static method named `colorToByte`, which takes a single parameter of type `Color` named `color` and returns a value of type `int`, representing compressed color information stored in one byte.
  - Retrieve the red component from the `color` object and reduce its precision to three bits:
    * Call the method `getRed` on the `color` object to obtain the red component value in the range 0–255.
    * Shift this value right by 5 bits to retain only the three most significant bits, and assign the result to the local variable `r` of type `int`.
  - Retrieve the green component from the `color` object and reduce its precision to three bits:
    * Call the method `getGreen` on the `color` object to obtain the green component value in the range 0–255.
    * Shift this value right by 5 bits to retain only the three most significant bits, and assign the result to the local variable `g` of type `int`.
  - Retrieve the blue component from the `color` object and reduce its precision to two bits:
    * Call the method `getBlue` on the `color` object to obtain the blue component value in the range 0–255.
    * Shift this value right by 6 bits to retain the two most significant bit positions, and assign the result to the local variable `b` of type `int`.
  - Build a single byte describing the color from the reduced components and return it:
    * Shift the value `r` left by 5 bits to place the three bits of red in the most significant positions of the byte.
    * Shift the value `g` left by 2 bits to place the three bits of green in the middle positions.
    * Leave the value `b` unshifted so that its two bits occupy the least significant positions of the byte.
    * Combine all three parts using the bitwise OR operation and return the resulting value.

- Declare a public static method named `byteToColor`, which takes a single parameter of type `int` named `value`, representing an encoded color byte, and returns an object of type `Color` reconstructing an approximate color value.

  - Extract from the parameter `value` the red component in reduced three-bit form:
    * Shift the value `value` right by 5 bits, moving the red bits to the least significant positions.
    * Apply the mask `0b111` to retain only three bits, and assign the result to the local variable `r` of type `int`.
  - Extract from `value` the green component in reduced three-bit form:
    * Shift the value `value` right by 2 bits, moving the green bits to the least significant positions.
    * Apply the mask `0b111` to retain the three least significant bits, and assign the result to the local variable `g` of type `int`.
  - Extract from `value` the blue component in reduced two-bit form:
    * Apply the mask `0b11` to retain only the two least significant bits.
    * Assign the result to the local variable `b` of type `int`, interpreting it as a two-bit blue component.
  - Rescale each reduced component to the full 0–255 range:
    * Compute the value `red` by multiplying `r` by 255 and dividing by 7.
    * Compute the value `green` analogously by multiplying `g` by 255 and dividing by 7.
    * Compute the value `blue` by multiplying `b` by 255 and dividing by 3.
  - Create a new color object from the rescaled components and return it:
    * Construct a new `Color` object, passing `red`, `green`, and `blue` to the constructor.
    * Return the created `Color` object, allowing the caller to obtain the color corresponding to the decompressed input value.
- Verify the correctness of the implemented methods using the attached test class `PositionAndColorTest.java`.
- Present positive test results to the instructor.